Different Data Models in Database Management Systems

1 Introduction

A **data model** is a collection of concepts used to describe how data is stored, organized, connected, and manipulated in a database.

In simple terms, a data model defines:

- How data is represented
- How different data items are related
- How data can be accessed
- What rules or constraints are applied to the data

For example, information about students may be stored as:

- Rows and columns in a table
- Objects with properties
- Documents in JSON format
- Nodes connected through relationships

The major data models are:

1. Hierarchical Data Model
2. Network Data Model
3. Relational Data Model
4. Entity-Relationship Data Model
5. Object-Oriented Data Model
6. Object-Relational Data Model

7. Document Data Model
8. Key-Value Data Model
9. Column-Family Data Model
10. Graph Data Model

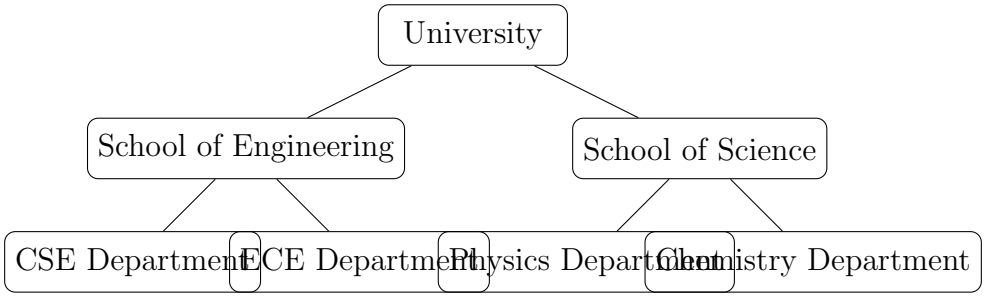
2 Hierarchical Data Model

The **hierarchical data model** organizes data in the form of a tree.

It follows a **parent-child relationship**. A parent may have many children, but a child normally has only one parent.

Example

Consider the organizational structure of a university:



Here, the university is the root node. Schools are child nodes, and departments are children of the schools.

Characteristics

- Data is arranged in a tree structure.
- It supports one-to-many relationships.
- Each child has only one parent.
- Data is accessed by following a fixed path.

Advantages

- Simple and easy to understand
- Fast access when the hierarchy is already known
- Suitable for naturally hierarchical data

Disadvantages

- Difficult to represent many-to-many relationships
- Structure is rigid
- Changes in the hierarchy may require database redesign

Applications

- File systems
- Organization structures
- XML documents
- Domain Name System

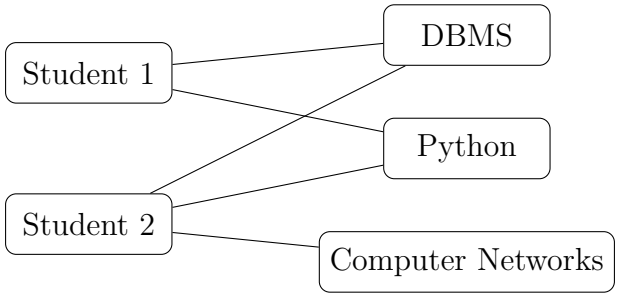
3 Network Data Model

The **network data model** organizes data as a graph-like structure.

Unlike the hierarchical model, a child record can have more than one parent. Therefore, it supports both one-to-many and many-to-many relationships.

Example

A student can enrol in multiple courses, and one course can have multiple students.



Characteristics

- Data is represented using records and links.
- A child may have multiple parents.
- It supports complex relationships.
- Data is accessed by navigating through links.

Advantages

- Supports many-to-many relationships
- More flexible than the hierarchical model
- Efficient for highly connected data

Disadvantages

- Complex database design
- Difficult to maintain
- Users must understand the access paths

4 Relational Data Model

The **relational data model** stores data in the form of tables, also called relations. Each table contains:

- Rows, called records or tuples
- Columns, called fields or attributes

Tables are connected through keys.

Example: Student Table

Student ID	Name	Department	Age
101	Rahul	CSE	20
102	Priya	ECE	21
103	Aman	CSE	19

In this table, `Student ID` may be used as the primary key.

Example: Course Table

Course ID	Course Name	Credits
C01	Database Management Systems	4
C02	Python Programming	3
C03	Computer Networks	4

Characteristics

- Data is stored in tables.
- Every row is uniquely identified by a primary key.
- Tables are connected using foreign keys.
- SQL is commonly used to access and manipulate data.

Example SQL Query

```
SELECT Name, Department
FROM Student
WHERE Department = 'CSE';
```

Advantages

- Simple tabular representation
- Easy to query using SQL
- Reduces data duplication through normalization
- Maintains data consistency and integrity
- Widely used in business applications

Disadvantages

- Complex joins may be slow
- Not suitable for highly unstructured data
- Difficult to represent deeply connected data

Examples of Relational Databases

- MySQL
- PostgreSQL
- Oracle Database
- Microsoft SQL Server
- SQLite

5 Entity-Relationship Data Model

The **Entity-Relationship model**, or **ER model**, is a conceptual data model used to design a database before creating the actual tables.

It represents data using:

- Entities
- Attributes
- Relationships

Entity

An entity is a real-world object about which data is stored.

Examples include:

- Student
- Teacher
- Course
- Department

Attribute

An attribute describes an entity.

For example, a Student entity may have the following attributes:

- Student ID
- Name
- Age
- Email

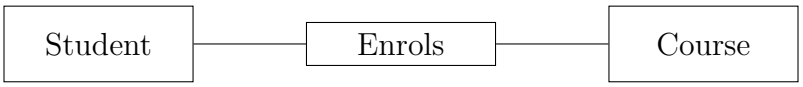
Relationship

A relationship shows how two entities are associated.

For example:

$$Student \longrightarrow Enrolsin \longrightarrow Course$$

Simple ER Diagram



Advantages

- Easy to understand
- Helps in database planning
- Clearly represents entities and relationships
- Can be converted into relational tables

Limitation

The ER model is mainly used for database design. It does not directly describe the physical storage of data.

6 Object-Oriented Data Model

The **object-oriented data model** stores data as objects, similar to object-oriented programming languages.

An object contains:

- Data or attributes
- Methods or functions

Example

A Student object may be represented as:

```
Student {
    studentID: 101,
    name: "Rahul",
    department: "CSE",

    displayDetails() {
        print(studentID, name, department);
    }
}
```

Characteristics

- Supports objects and classes
- Supports inheritance
- Supports encapsulation
- Supports polymorphism
- Can store complex data types

Advantages

- Suitable for complex applications
- Can store multimedia and scientific data
- Matches object-oriented programming languages
- Reduces the difference between program objects and database records

Disadvantages

- More complex than relational databases
- Lacks a universally accepted query language
- Less commonly used in general business applications

Applications

- Computer-aided design
- Multimedia systems
- Geographic information systems
- Scientific applications

7 Object-Relational Data Model

The **object-relational data model** combines the features of the relational model and the object-oriented model.

It stores data in tables but also supports complex data types and object-oriented concepts.

Example

A relational table may contain a user-defined address data type:

```
CREATE TYPE AddressType AS (
    city VARCHAR(50),
    state VARCHAR(50),
    postal_code VARCHAR(10)
);
```

A Student table may then use this address type:

```
CREATE TABLE Student (
    student_id INTEGER PRIMARY KEY,
    name VARCHAR(100),
    address AddressType
);
```

Advantages

- Supports tables and objects
- Handles complex data types
- Maintains compatibility with SQL
- Suitable for advanced applications

Examples

- PostgreSQL
- Oracle Database

8 Document Data Model

The **document data model** stores data as documents.

Documents are commonly represented using:

- JSON
- BSON
- XML

Each document may have a different structure.

Example JSON Document

```
{
    "_id": 101,
    "name": "Rahul",
    "department": "CSE",
    "subjects": [
        "DBMS",
        "Python",
        "Computer Networks"
    ],
    "address": {
        "city": "Kolkata",
        "state": "West Bengal"
    }
}
```

Characteristics

- Data is stored as self-contained documents.
- Documents may contain nested objects and arrays.
- A fixed table structure is not always required.
- Documents within the same collection may have different fields.

Advantages

- Flexible schema
- Suitable for semi-structured data
- Easy to store nested data
- Convenient for web and mobile applications

Disadvantages

- Data duplication may occur
- Complex relationships may be difficult to manage
- Transactions across many documents may be complicated

Examples

- MongoDB
- CouchDB
- Firebase Firestore

9 Key-Value Data Model

The **key-value data model** stores data as pairs consisting of a unique key and its corresponding value.

$$Key \longrightarrow Value$$

Example

Key	Value
student:101	Rahul
student:102	Priya
course:C01	Database Management Systems

A value may also contain a JSON object:

```
Key: student:101

Value:
{
    "name": "Rahul",
    "department": "CSE",
    "age": 20
}
```

Characteristics

- Every value is identified by a unique key.
- Data is usually retrieved using the key.
- The database does not necessarily understand the internal structure of the value.

Advantages

- Very fast data retrieval
- Simple structure
- Highly scalable
- Suitable for caching and session management

Disadvantages

- Limited querying capabilities
- Difficult to search using properties inside values
- Not suitable for complex relationships

Examples

- Redis
- Amazon DynamoDB
- Riak
- Valkey

10 Column-Family Data Model

The **column-family data model** stores data in rows and columns, but the columns are grouped into column families.

Unlike a relational table, different rows may contain different columns.

Example

Row Key	Name	Department	Phone
101	Rahul	CSE	9876543210
102	Priya	ECE	—
103	Aman	—	9123456780

Here, all rows do not need to contain exactly the same columns.

Characteristics

- Data is organized using row keys.
- Related columns are grouped into column families.
- Different rows may have different columns.
- It is designed for distributed storage.

Advantages

- Suitable for very large datasets
- Highly scalable
- Efficient for distributed systems
- Handles sparse data effectively

Disadvantages

- More difficult to design than relational tables
- Limited support for joins
- Query patterns should generally be planned in advance

Examples

- Apache Cassandra
- Apache HBase
- Google Bigtable

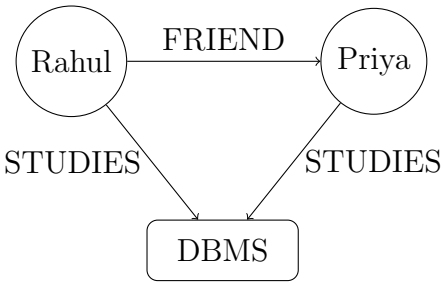
11 Graph Data Model

The **graph data model** represents data using nodes, edges, and properties.

- A **node** represents an entity.
- An **edge** represents a relationship.
- A **property** stores information about a node or relationship.

Example

Consider a social networking application:



Characteristics

- Data is represented as nodes and relationships.
- Relationships are directly stored.
- It is suitable for highly connected data.
- Traversing relationships is efficient.

Advantages

- Efficient for relationship-based queries
- Flexible structure
- Suitable for complex networks
- Avoids expensive relational joins

Disadvantages

- Not suitable for every type of application
- Requires knowledge of graph query languages
- May be more complex for simple tabular data

Applications

- Social networks
- Recommendation systems
- Fraud detection
- Route and navigation systems
- Knowledge graphs

Examples

- Neo4j
- Amazon Neptune
- ArangoDB

12 Comparison of Different Data Models

Data Model	Structure	Suitable For	Example Database
Hierarchical	Tree structure	Organization hierarchy and file systems	IBM IMS
Network	Records connected through links	Complex many-to-many relationships	Integrated Data Store
Relational	Tables containing rows and columns	Banking, education, inventory and business systems	MySQL, PostgreSQL
Entity-Relationship	Entities, attributes and relationships	Conceptual database design	Used during database planning
Object-Oriented	Objects and classes	Multimedia, engineering and scientific applications	ObjectDB
Object-Relational	Tables with object-oriented features	Complex applications requiring SQL	PostgreSQL, Oracle
Document	JSON, BSON or XML documents	Web applications and semi-structured data	MongoDB, CouchDB
Key-Value	Unique key and associated value	Caching, sessions and fast lookups	Redis, Valkey
Column-Family	Rows with groups of flexible columns	Large-scale distributed data	Cassandra, HBase
Graph	Nodes, edges and properties	Social networks and relationship-based applications	Neo4j

13 Relational and Non-Relational Data Models

Data models may also be broadly classified as relational and non-relational.

Relational Model

The relational model stores data in fixed tables with rows and columns.

Examples:

- MySQL
- PostgreSQL
- Oracle

Non-Relational Model

Non-relational databases are commonly called **NoSQL databases**.

Major NoSQL data models include:

- Document model
- Key-value model
- Column-family model
- Graph model

Comparison

Feature	**Relational Model**	**Non-Relational Model**
Structure	Tables with rows and columns	Documents, key-value pairs, graphs or column families
Schema	Usually fixed and predefined	Often flexible or dynamic
Relationships	Managed using primary and foreign keys	Managed using nesting, links or graph edges
Query language	Usually SQL	Depends on the database
Scaling	Traditionally vertical scaling	Commonly designed for horizontal scaling
Best suited for	Structured data and transactions	Large-scale, flexible or highly connected data
Examples	MySQL and PostgreSQL	MongoDB, Redis, Cassandra and Neo4j

14 Example: Representing the Same Data Using Different Models

Suppose we want to store the following information:

> Rahul is a CSE student. He studies DBMS and Python.

Relational Representation

Student Table

Student ID	Name	Department
101	Rahul	CSE

Course Table

Course ID	Course Name
C01	DBMS
C02	Python

Enrolment Table

Student ID	Course ID
101	C01
101	C02

Document Representation

```
{
    "student_id": 101,
    "name": "Rahul",
    "department": "CSE",
    "courses": ["DBMS", "Python"]
}
```

Key-Value Representation

```
Key: student:101
```

```
Value:
{
    "name": "Rahul",
    "department": "CSE",
    "courses": ["DBMS", "Python"]
}
```

Graph Representation

$$Rahul STUDIES DBMS$$

$$Rahul STUDIES Python$$

15 How to Select a Data Model

The selection of a data model depends on the requirements of the application.

- Use the **relational model** when data is structured and transactions are important.
- Use the **document model** when data contains nested structures or the schema changes frequently.
- Use the **key-value model** when extremely fast access using a key is required.
- Use the **column-family model** for large-scale distributed and sparse datasets.
- Use the **graph model** when relationships between data items are the main focus.
- Use the **hierarchical model** when data naturally follows a tree structure.

16 Object Store

An **object store** stores data as individual objects. An object usually contains the actual data, metadata describing the data, and a unique identifier.

Unlike a traditional file system, objects are generally stored in a flat namespace rather than inside a hierarchy of folders.

Example

An image stored in an object storage system may contain:

```
Object Key: student-images/101.jpg

Data:
Binary image content

Metadata:
{
    "student_id": 101,
    "file_type": "JPEG",
    "uploaded_by": "Administrator"
}
```

Characteristics

- Stores files and binary data as objects
- Each object has a unique key
- Objects may contain descriptive metadata
- Designed for large-scale distributed storage

Advantages

- Highly scalable
- Suitable for large files
- Useful for backups, videos, images and documents
- Easy to access through web-based APIs

Limitations

- Not suitable for complex relational queries
- Objects are usually retrieved using their keys
- Frequent modification of part of a large object may be inefficient

Examples

- Amazon S3
- Google Cloud Storage
- Azure Blob Storage
- MinIO

17 Multimodel Database

A **multimodel database** supports more than one data model within the same database system.

For example, a single database may support document, graph and key-value representations.

Example

A university application may use:

- Documents for student profiles
- Graphs for student-course relationships
- Key-value pairs for login sessions

All these forms of data may be managed by one multimodel database.

Characteristics

- Supports multiple data models
- Provides a common database environment
- Reduces the need for separate database systems
- Allows different query methods for different data structures

Advantages

- Flexible database design
- Supports different application requirements
- Reduces integration between multiple databases
- Suitable for complex modern applications

Limitations

- More complex to administer
- Performance may vary between supported models
- Requires knowledge of multiple query methods

Examples

- ArangoDB
- OrientDB
- Azure Cosmos DB

18 Time-Series Data Model

The **time-series data model** is designed for data recorded continuously over time. Each record normally contains:

- A timestamp
- A measured value
- Optional labels or tags

Example

An IoT temperature sensor may generate the following data:

Timestamp	Device ID	Temperature
2026-07-29 09:00:00	D01	28.5
2026-07-29 09:01:00	D01	28.7
2026-07-29 09:02:00	D01	29.0

Characteristics

- Data is arranged according to time
- Optimized for timestamp-based queries
- Supports continuous insertion of records
- Commonly used for measurements, logs and events

Advantages

- Fast storage of continuously generated data
- Efficient time-range queries
- Suitable for monitoring and trend analysis
- Supports aggregation over time intervals

Limitations

- Mainly designed for time-dependent data
- Complex relationships may require another database model
- Large data volumes may require retention policies

Examples

- InfluxDB
- TimescaleDB
- OpenTSDB
- Prometheus

19 XML Data Model

The **XML data model** stores data in the form of XML documents.

XML represents data using nested tags and attributes.

Example

```
<student id="101">
    <name>Rahul</name>
    <department>CSE</department>
    <courses>
        <course>DBMS</course>
        <course>Python</course>
    </courses>
</student>
```

Characteristics

- Data is represented using tags
- Supports hierarchical and nested structures
- Documents may be validated using XML Schema
- XML-specific query languages may be used

Advantages

- Self-descriptive structure
- Suitable for data exchange
- Supports nested and hierarchical information
- Platform-independent representation

Limitations

- XML documents are often verbose
- Storage requirements may be high
- Parsing may be slower than simpler formats

Examples

- BaseX
- eXist-db
- MarkLogic

20 RDF Store or Triplestore

An **RDF store**, also called a **triplestore**, stores data as subject-predicate-object statements.

The basic structure is:

$$Subject \longrightarrow Predicate \longrightarrow Object$$

Example

$$RahulstudiesDBMS$$

This statement can be represented as:

```
Subject:   Rahul
Predicate: studies
Object:    DBMS
```

Another example is:

```
Rahul   belongsTo   CSE
DBMS    taughtBy    DrSharma
```

Characteristics

- Stores data as triples
- Represents semantic relationships
- Supports linked and interconnected data
- Commonly queried using SPARQL

Advantages

- Suitable for semantic data
- Flexible relationship representation
- Supports linked-data applications
- Useful for knowledge graphs

Limitations

- More complex than ordinary tabular storage
- Requires knowledge of RDF and SPARQL
- May not be suitable for ordinary transaction processing

Examples

- Virtuoso
- Stardog
- Apache Jena
- GraphDB

21 Search-Oriented Data Model

A **search-oriented data model** is designed for indexing and retrieving textual information efficiently.

The data is usually stored as searchable documents. Important words and terms are recorded in an index so that documents can be found quickly.

Example Document

```
{
    "student_id": 101,
    "name": "Rahul",
    "department": "CSE",
    "research_interest": "Artificial Intelligence and IoT"
}
```

A search query such as:

`Artificial Intelligence`

can retrieve all documents containing related terms.

Characteristics

- Uses indexes for fast text searching
- Stores searchable documents
- Supports keyword and full-text search
- Often provides relevance-based ranking

Advantages

- Fast full-text searching
- Supports filtering and ranking
- Suitable for logs and document search
- Can search large collections of textual data

Limitations

- Not normally used as the main transactional database
- Indexed data may require additional storage
- Search indexes must be updated when source data changes

Examples

- Elasticsearch
- Apache Solr
- OpenSearch

22 Conclusion

A data model determines how information is represented and connected inside a database. Different data models are suitable for different application requirements.

The relational model is suitable for structured data and transactions, while document, key-value, column-family, and graph models are suitable for flexible, distributed, or relationship-oriented applications.

Therefore, no single data model is best for every application. The correct model should be selected based on the data structure, relationships, query requirements, scalability, and consistency needs.